Centre de recherche sur les transports (C.R.T.)
Centre for Research on Transportation

CRT-2005-08

NEW REFINEMENTS FOR THE SOLUTION OF VEHICLE ROUTING PROBLEMS WITH BRANCH AND PRICE

by

Dominique Feillet
Michel Gendreau
Louis-Martin Rousseau

April 2005

Université de Montréal
C.P. 6128, succursale Centre-ville, Montréal QC H3C 3J7 Canada
Téléphone : (514) 343-7575 / Télécopieur : (514) 343-7121

crt@crt.umontreal.ca
www.crt.umontreal.ca

HEC ÉCOLE POLYTECHNIQUE MONTRÉAL Université de Montréal

105-03979/

PUBLICATION
CRT-2005-08

NEW REFINEMENTS FOR THE SOLUTION OF VEHICLE ROUTING PROBLEMS WITH BRANCH AND PRICE

by

Dominique Feillet[1]
Michel Gendreau[2]
Louis-Martin Rousseau[3]

The research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), and by the Fonds québécois de la recherche sur la nature et les technologies (FQRNT).

[1] Laboratoire d'Informatique d'Avignon, 339 Chemin des Meinajariés, BP 1228, 84911 Avignon, France (dominique.feillet@univ-avignon.fr)

[2] Département d'informatique et recherche opérationnelle and Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7 (michelg@crt.umontreal.ca)

[3] Département de mathématiques et génie industriel, École Polytechnique de Montréal, C. P. 6079, succursale Centre-ville, Montréal, Canada H3C 3A7 and Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7 (louism@crt.umontreal.ca)

Département d'informatique et de recherche opérationnelle – Publication #1260

Dépôt légal – Bibliothèque nationale du Québec
Bibliothèque nationale du Canada, 2005

Abstract

Column generation is a well-known mathematical programming technique based on two components: a master problem, that selects optimal columns (variables) in a restricted pool of columns, and a subproblem that feeds this pool with potentially good columns until an optimality criterion is met. Embedded in Branch and Price algorithms, this solution approach proved to be very efficient in the context of numerous vehicle routing problems, where columns are feasible vehicle routes. The subproblem is then generally expressed as a shortest path problem with resource constraints, that can be solved efficiently using dynamic programming. In this paper, we propose some new refinements to improve the capabilities of column generation approaches in this context, with a focus on the subproblem phase. For sake of simplicity, we restrict our study to the case of the Vehicle Routing Problem with Time Windows. We first introduce the notion of Limited Discrepancy Search known in Constraint Programming to the field of Dynamic Programming; we also discuss how the state graph of dynamic programming can be manipulated in order to simulate local search during label extension; finally, we present some lower bounds that allow removing a substantial amount of labels during the search. Computational results demonstrate the considerable impact of these refinements in terms of computing time.

Keywords: column generation; branch and price; vehicle routing.

Résumé

La génération de colonnes est une approche bien connue en programmation mathématique. Elle repose sur deux composantes: un problème maître qui choisit les colonnes (variables) optimales parmi un ensemble restreint de colonnes, et un sous-problème qui nourrit cet ensemble avec des colonnes prometteuses jusqu'à ce qu'un certain critère d'optimalité soit satisfait. Intégrée dans des algorithmes de "Branch and Price", cette approche s'est montrée très efficace pour la résolution de nombreux problèmes de confection de tournées de véhicules dans lesquels les colonnes correspondent à des tournées réalisables. Le sous-problème est alors généralement posé sous la forme d'un problème de chemin le plus court avec contraintes de ressources qui peut être résolu efficacement par des techniques de programmation dynamique. Dans cet article, nous proposons divers raffinements qui visent à rendre les approches de génération de colonnes plus efficaces dans ce contexte, avec une emphase sur le sous-problème. Afin de simplifier la discussion, nous nous limitons au problème de confection de tournées de véhicules avec fenêtres de temps. Nous appliquons tout d'abord à la programmation dynamique la notion de"Limited Discrepancy Search" issue de la programmation avec contraintes. Nous montrons ensuite comment le graphe d'états de la programmation dynamique peut être manipulé pour simuler l'effet d'une recherche locale au moment de l'extension des étiquettes. Finalement, nous présentons des bornes inférieures qui permettent d'éliminer un nombre important d'étiquettes durant la recherche. Les expérimentations numériques démontrent l'impact considérable de ces raffinements sur les temps de calcul.

Mots-clés: génération de colonnes; branch and price; confection de tournées de véhicules.

Introduction

Vehicle routing problems are widely present in today's industries, ranging from distribution problems to fleet management. They account for a significant portion of the operational costs of many companies. Operations research techniques have been used with success in many situations for reducing such costs. Even if most real instances of vehicle routing problems are solved with heuristic methods, the desire to produce optimal solutions has given rise to a prolific research area. Through the years, the Branch and Price methodology proved to be a cornerstone for the exact solution of many vehicle routing problems. This includes routing problems with time windows (Desrochers *et al.* 1992), backhauls (Gélinas *et al.* 1995) or pick-up and delivery (Sol 1994) to mention only a few.

Among these problems, the Vehicle Routing Problem with Time Windows (VRPTW) can be described as follows. Given a set of customers, a set of vehicles, and a depot, the VRPTW is to find a set of routes of minimal total length, starting and ending at the depot, such that each customer is visited by exactly one vehicle to satisfy a specific demand. Customer visit has to respect a requested time window. A vehicle can wait in case of early arrival, but late arrival is not permitted. In connection with customer demands, a capacity constraint restricts the load that can be carried by a vehicle.

In 1992, Desrochers *et al.* (1992) published a seminal paper for the solution of the VRPTW with a Branch and Price procedure. The procedure was evaluated on a large set of instances, involving up to 100 customers, but only managed to solve a limited subset of these instances. Since then, several authors have adapted this approach, introducing some more evolved concepts and permitting to solve more and more instances. Even so, many of these instances still remain open today while many can only be solved with undue computing times.

The purpose of this paper is to propose some new refinements to the Branch and Price methodology, helping in the solution of difficult instances. For sake of simplicity, we restrict our study to the case of the VRPTW. Even so, many of these refinements could easily be transposed for the solution of other vehicle routing problems.

Section 1 will introduce more accurately the VRPTW and will review Branch and Price approaches for its solution. Section 2 will describe the refinements and the algorithm that we propose. Thorough computational experiments first evaluating the algorithm and then the different refinements separately will be provided in Section 3.

1 Branch and Price methodology for the VRPTW

1.1 Formal description of the VRPTW

The VRPTW is defined on a network $G = (V, A)$, where $V = \{v_0, \ldots, v_n\}$ is the set of nodes and A is the set of arcs. Vertex v_0 is a special node called the depot, vertices v_1 to v_n represent customers. A cost c_{ij} and a travel time t_{ij} are defined for every arc $(v_i, v_j) \in A$. Every customer $v_i \in V \setminus \{v_0\}$ has a positive demand d_i, a time window $[a_i, b_i]$ and a positive service time s_i. A fleet of K vehicles of capacity Q is available for servicing the customers. Vehicles must begin and end their routes at the depot within a time horizon $[a_0, b_0]$. The cumulated demand of customers visited by a route is limited by Q. The service of a customer has to start within its time window, but a vehicle is allowed to arrive earlier and to wait. The VRPTW consists in finding a minimum cost set of routes visiting exactly once each customer with respect to the previous constraints.

In the following, we make these additional common assumptions: the cost and the travel time matrices are supposed to be equal, nonnegative and to satisfy the triangle inequality. For sake of simplicity, we also define $d_0 = 0$ and $s_0 = 0$.

The VRPTW can then be described with the following model:

$$\text{minimize} \sum_{1 \leq k \leq K} \sum_{(v_i, v_j) \in A} c_{ij} x_{ij}^k \tag{1}$$

subject to

$$\sum_{\{v_j \in V | (v_i,v_j) \in A\}} x_{ij}^k - \sum_{\{v_j \in V | (v_j,v_i) \in A\}} x_{ji}^k = 0 \qquad (v_i \in V, 1 \leq k \leq K), \tag{2}$$

$$\sum_{\{v_i \in V | (v_0,v_i) \in A\}} x_{0i}^k \leq 1 \qquad (1 \leq k \leq K), \tag{3}$$

$$\sum_{1 \leq k \leq K} \sum_{\{v_j \in V | (v_i,v_j) \in A\}} x_{ij}^k = 1 \qquad (v_i \in V \setminus \{v_0\}), \tag{4}$$

$$\sum_{(v_i,v_j) \in A} d_i x_{ij}^k \leq Q \qquad (1 \leq k \leq K), \tag{5}$$

$$s_i^k + s_i + c_{ij} - s_j^k + M x_{ij}^k \leq M \qquad ((v_i,v_j) \in A, v_j \neq v_0, 1 \leq k \leq K), \tag{6}$$

$$s_i^k + s_i + c_{i0} - b_0 + M x_{i0}^k \leq M \qquad ((v_i,v_0) \in A, 1 \leq k \leq K), \tag{7}$$

$$a_i \leq s_i^k \leq b_i \qquad (v_i \in V, 1 \leq k \leq K), \tag{8}$$

$$x_{ij}^k \in \{0,1\} \qquad ((v_i,v_j) \in A, 1 \leq k \leq K), \tag{9}$$

$$s_i^k \geq 0 \qquad (v_i \in V, 1 \leq k \leq K), \tag{10}$$

where x_{ij}^k and s_i^k are decision variables and M is a large number. Variables x_{ij}^k indicate whether arc (v_i, v_j) is used by vehicle k or not. For a customer v_i visited by a vehicle k_1, $s_i^{k_1}$ is the starting time of service, while s_i^k is meaningless for $k \neq k_1$. For the depot, s_0^k is the departure time of vehicle k.

Constraints (2)-(3) define the route structure for the vehicles. Constraints (4) enforce the visit of every customer. Constraints (5) and constraints (6)-(8) respectively concern vehicle capacity and time windows.

1.2 Branch and Price methodology for the VRPTW

In this section, we describe the principles of Branch and Price algorithms for the VRPTW and we present the most important works on the subject.

The motivation for using a Branch and Price technique here is that the linear relaxation of model (1)-(10) is very weak. Hence, this model cannot be used directly with a Branch and Bound approach, except when fairly small instances are considered. To circumvent this difficulty, Branch and Price methods rely on a different model having a better linear relaxation. This new model can be obtained through a Dantzig-Wolfe decomposition from (1)-(10) as detailed in Desrosiers *et al.* (1992). Before presenting this model, we have to introduce some new notation. Let $\Omega = \{r_1, \ldots, r_{|\Omega|}\}$ be the set of feasible vehicle routes, i.e., the set of paths in G issued from the depot, going to the depot, satisfying capacity and time window constraints and visiting at most once each customer. Let c_k be the cost of route $r_k \in \Omega$. Let $a_{ik} = 1$ if route $r_k \in \Omega$ visits customer v_i and 0 otherwise. The new model for the VRPTW is then:

$$\text{minimize} \sum_{r_k \in \Omega} c_k x_k \tag{11}$$

subject to

$$\sum_{r_k \in \Omega} a_{ik} x_k \geq 1 \qquad (v_i \in V \setminus \{v_0\}), \tag{12}$$

$$\sum_{r_k \in \Omega} x_k \leq K, \tag{13}$$

$$x_k \in \{0,1\} \qquad (r_k \in \Omega). \tag{14}$$

In this model, decision variables x_k indicate whether a route r_k is used in the solution or not. Constraint (13) limits the number of vehicles used. Constraints (12) enforce that each customer is

visited at least once. Note that these constraints are not formulated as $\sum_{r_k \in \Omega} a_{ik} x_k = 1$ for technical reasons to be explained later and because the triangle inequality ensures that there exists an optimal solution of (11)-(14) visiting each customer exactly once.

Although having a better linear relaxation value than (1)-(10), this model is not tractable with a standard Branch and Bound approach. This stems from the fact that the size of the set Ω grows exponentially with instance size. Indeed, the linear programs that would be used to evaluate search tree nodes would contain too many variables to be solved in a classical manner. This evaluation can however be tackled with a column generation technique, which transforms Branch and Bound into Branch and Price.

Column generation follows the subsequent principle. In the following, we call Master Problem (MP) the linear relaxation of (1)-(10). We introduce $MP(\Omega_1)$, the restriction of the Master Problem MP to a subset of variables $\Omega_1 \subset \Omega$. $MP(\Omega_1)$ is called the Restricted Master Problem. Let also $D(\Omega_1)$ be the dual program of $MP(\Omega_1)$. Note that MP is then identically $MP(\Omega)$ and that $D(\Omega)$ is the dual program of MP. The optimal solution of $MP(\Omega_1)$, with the simplex algorithm for instance, provides an optimal solution λ^* for $D(\Omega_1)$. This solution is also a solution of $D(\Omega)$, but it is not necessarily feasible. When every constraint deriving from the routes of of $\Omega \setminus \Omega_1$ is satisfied, the solution λ^* is feasible for $D(\Omega)$, and therefore optimal, since $D(\Omega)$ is more constrained than $D(\Omega_1)$. When one or several constraints deriving from the routes of $\Omega \setminus \Omega_1$ are violated, the principle of the column generation method is to identify one or several of these constraints, with the help of a subproblem, in order to integrate the corresponding variables in the set Ω_1. Thus, solving alternately $MP(\Omega_1)$ and the subproblem allows to converge toward dual feasibility. The algorithm terminates when the subproblem solution attests that there are no more violated constraints and that therefore the current dual solution is feasible for $D(\Omega)$.

In our situation, dual constraints are of the form $\sum_{v_i \in V \setminus \{v_0\}} a_i^k \lambda_i + \lambda_0 \leq c_k$, where λ_i is the nonnegative dual variable associated with the visit of customer v_i (constraints (12)) and λ_0 is the nonpositive dual variable associated with the fleet size constraint (13). Note that formulating constraints (4) $\sum_{r_k \in \Omega} a_{ik} x_k = 1$ would have led to free variables λ_i, which would have complicated the convergence of the algorithm.

The purpose of the subproblem is finally to find routes $r_k \in \Omega$ such that

$$c_k - \sum_{v_i \in V \setminus \{v_0\}} a_i^k \lambda_i - \lambda_0 < 0.$$

It consists equivalently of columns with a negative reduced cost in $MP(\Omega)$, when the basic solution is the optimal solution of $MP(\Omega_1)$. In the following, we call these columns routes with a negative reduced cost. Using the notation $\delta_{ij}^k = 1$ when (v_i, v_j) is included in r_k and $\delta_{ij}^k = 0$ otherwise, this condition can be expressed as:

$$\sum_{(v_i, v_j) \in A} \delta_{ij}^k (c_{ij} - \lambda_i) < 0.$$

This expression shows that the subproblem reduces to an elementary shortest path problem with resource constraints (ESPPRC). One has to find low cost elementary paths from the depot to the depot, satisfying capacity and time constraints. A solution procedure based on dynamic programming for this problem is proposed in Feillet *et al.* (2004). Dynamic programming is particularly well adapted to this context because it computes a set of Pareto optimal paths and might provide MP with several columns at a time. Here, we give a brief description of the algorithm used to solve the subproblem. The algorithm is an extension of the classical Bellman's algorithm. The principle is to associate with each possible partial path a label and to extend these labels checking resource constraints until the best feasible paths are obtained. Dominance rules are used to compare partial paths arriving at a same location and to discard some of them. But, unlike Bellman's algorithm when no resources are considered, each vertex of the graph can maintain a large number of labels since the comparison of two labels takes into account their consumption level for each resource. A common practice is to avoid computing the complete Pareto optimal path set by stopping the subproblem solution algorithm prematurely when a sufficient number of good columns has been found.

Another important point of the Branch and Price method concerns the branching scheme. Instead of branching on MP variables, it is generally preferred to branch on variables x_{ij}^k from the original formulation (1)-(10). It is fairly easy to see that when MP has a fractional optimal solution, there exists at least one arc (v_i, v_j) that is traversed a fractional number of times f_{ij} with $0 < f_{ij} < 1$ (where $f_{ij} = \sum_{r_k \in \Omega_i} \delta_{ij}^k x_k$). It is then possible to derive two branches: one branch where v_j cannot follow v_i, the other where a vehicle visiting v_i necessarily goes immediately to v_j. These two rules can very easily be transposed to MP: in both cases, inadequate columns are just set to 0. For the subproblem, it is not much more complicated. In the first case, arc (v_i, v_j) is simply removed. In the second case, any arc (v_i, v_l) with $v_l \neq v_j$ and any arc (v_l, v_j) with $v_l \neq v_i$ is removed.

1.3 Evolved Branch and Price implementations

In this section we review several techniques that have been introduced within this general scheme and that have allowed to significantly increase its efficiency.

In the first implementation of Branch and Price for the VRPTW, Desrochers *et al.* (1992) transformed (11)-(14) to obtain a more tractable subproblem. This transformation takes advantage on the fact that the Elementary Shortest Path Problem with Resource Constraints, though NP-hard in the strong sense, admits a pseudo-polynomial algorithm when the elementary path condition is removed. Thereby, Desrochers *et al.* (1992) propose the two following modifications:

- Ω is enlarged to include non elementary paths,
- a_i^k becomes the number of times vertex v_i is visited in route r_k.

With these two simple changes, the model remains valid and the subproblem is changed into a (non elementary) shortest path problem with resource constraints (SPPRC). It can be solved with a dynamic programming similar to the one mentioned above. Actually, Feillet *et al.*'s algorithm (2004) for the ESPPRC derives from this one. Even if the size of the subproblem state space increases with this modification, dominance rules are far more efficient and the global efficiency is improved. However, with this new formulation, the linear relaxation provides a weaker lower bound, which complicates the pruning of nodes during the tree search.

Actually, Desrochers *et al.* (1992) were more clever and forbade paths with 2-cycles, i.e., paths with cycles composed of two arcs. This condition can easily be added to the Master Problem and can be handled by the SPPRC dynamic programming solution algorithm with a low computational cost. Following this idea, Irnich (2001) and Irnich and Villeneuve (2003) study the removing of k-cycles. They evaluate it for $k = 3$ and $k = 4$ and conclude that the quality of the lower bound can come significantly closer to the ESPPRC-based bound, with a reasonable computing time. More recently, in addition to Feillet *et al.* (2004), Chabrier (2003) adapted efficiently the SPPRC dynamic programming solution algorithm to the ESPPRC situation.

Another approach proposed for increasing the quality of the lower bound is the addition of cuts. Kohl *et al.* (1999) introduced the so-called k-path cuts that are valid for the VRPTW. These cuts are of the form

$$\sum_{i \in V \setminus S} \sum_{j \in S} f_{ij} \geq k,$$

where f_{ij} is the flow on arc (v_i, v_j) as defined above and S is a vertex set whose demand cannot be satisfied with $k-1$ vehicles. These cuts can easily be integrated into the Master Problem and handled by the subproblem by only taking into account new dual variables on appropriate arcs. However, they are quite complicated to separate. Kohl *et al.* (1999) limit their study to k-path cuts with $k \leq 2$. The case $k = 1$ is very simple. When $k = 2$, an heuristic algorithm is proposed to find maximal sets S such that $\sum_{i \in V \setminus S} \sum_{j \in S} f_{ij} < 2$ and a Traveling Salesman Problem with Time Windows (TSPTW) solution algorithm is used to determine whether these sets can be visited by a single vehicle. The case $k = 3$ was addressed later by Cook and Rich (1999).

Finally, Kallehauge *et al.* [12] proposed to hybridize the Branch and Price scheme with Lagrangean relaxation. Column generation can be viewed as a decomposition scheme where the Master Problem and the subproblem communicate with the help of values λ_i, when these values are set to be the dual variables of the Master Problem. Lagrangean relaxation involves identical Master Problem and subproblem, communicating with values λ_i, but these values are guided by a different strategy. Kallehauge *et al.* [12] propose to use Lagrangean relaxation at the root node to improve the convergence of the method.

Beside these strategies, many computational tricks have been proposed to accelerate the solution process. Most of these tricks are summarized in Desaulniers *et al.* (2001). All these attempts have allowed solving new instances as will be recalled in section 3.

2 New Acceleration Techniques

The contribution of this paper lies in the following improvements that are incorporated in the subproblem solution algorithm. Note that contrary to usual implementations of column generation for routing problems, we only consider elementary routes in Ω, i.e., routes where customers are never visited more than once. Indeed, we aim to accelerate the subproblem solution, which should permit to maintain a good quality of lower bound. But, although these techniques are implemented in the context of elementary shortest paths, they can easily be transposed to a non-elementary context, unless the opposite is explicitly mentioned.

2.1 Limited Discrepancy Search

Limited Discrepancy Search (LDS), introduced by Harvey and Ginsberg (1995), is a well-known tree search method in Constraint Programming (CP). Given a heuristic that ranks all the possible branches at a given node of a search arborescence, a discrepancy is a branching decision that does not follow the heuristic recommendation. LDS trusts the heuristic by first exploring the branches that correspond to lowest discrepancy and by setting a limit on the maximum number of discrepancy allowed. Limited Discrepancy Search is thus a very effective technique that allows identifying rapidly good solutions by exploring the most promising region of the solution space. Furthermore, compared to a heuristic tree search that would only explore good branches, it has the advantage to allow some rare bad decisions during the search.

We embed the concept of LDS in Dynamic Programming (DP) to efficiently generate the most promising paths of negative value. LDS is implemented by adding a dimension to every label that indicates the number of discrepancies performed in order to generate that label. For each vertex of the original VRPTW, we define a set of good neighbors and a set of bad neighbors according to reduced cost considerations. A discrepancy is then counted each time a label is extended along an arc connecting a vertex to one of its bad neighbors. If the extension of a label to a particular vertex causes the discrepancy level of that label to exceed the current global maximum discrepancy level, the label is simply not extended. Figure 1 provides an illustration of this concept.

Although simple, this technique can be declined in many ways. In our implementation, we have proceeded as follows. Two parameters are defined for the size of the good neighbor set and for the number of discrepancies allowed (called DISC). While the size of the good neighbor set is fixed throughout the resolution, DISC is increased when the subproblem fails to find new routes. The ESPPRC is then solved again until routes are found or the value of DISC ensures that the search is complete, which closes the subproblem solution phase.

Good neighbors are the two neighbors with the minimum reduced cost value $c_{ij} - \lambda_i$, plus the depot. For sake of efficiency, lists of successors are sorted according to reduced cost at each new iteration of the column generation scheme (i.e., when dual values have changed). This permits to stop trying to extend a label toward its neighbors once an extension is discarded due to discrepancy.

When, following a fail, the ESPPRC is solved again with a higher value for DISC, former labels are preserved. DP then performs normally with a set of labels that can potentially be extended. The

Figure 1: LDS in Dynamic Programming: example of a label with one discrepancy.

DISC parameter is regulated as follows. After a fail, its value is doubled, unless the new value would exceed half of the maximal number of customers that a route can contain. In this case, it is fixed to this maximal number. Also, when a fail occurs with DISC equal to 0, DISC is set to 1. If DISC is maximal and the subproblem still fails to find new routes, the search is complete and the column generation is stopped. The maximal number of customers in routes is precomputed with the solution of two knapsack problems where items are customers and weights are respectively the demand and the minimum time consumed for visiting a customer (service time plus cost of the cheapest outgoing arc). Actually, for efficiency reasons, we only solve the linear relaxation of these knapsack problems.

A third parameter DISCMP is used to regulate LDS at the Master Problem level. This parameter defines the initial value of DISC for each call of the subproblem. At the first iteration of column generation, DISCMP is set to 0. Then, at each iteration of the column generation scheme, DISCMP is updated depending on the behavior of DISC during the previous subproblem solution phase. When DISC has increased, we expect that finding new routes was difficult and DISCMP is doubled (or set to 1 or to the maximum number of customers in routes similarly as above). On the contrary, if DISC did not need to be increased, the search is expected to be easier and DISCMP is halved.

2.2 Label Loading and Meta Extensions

The motivation behind these techniques is to use, when possible, the information about the "good" paths that have been previously identified. Indeed, once MP is solved, we already have in hand a number of routes whose reduced cost value is zero (namely, MP basic columns).

Label Loading (LL) consists of adding a set of labels to the graph before the DP search process is undertaken. This is very simple and has presumably been implemented in other DP algorithms addressing similar problems. Our implementation consists in selecting in Ω_1 all routes r for which $x_r > 0$. We then traverse each of these routes while generating the label associated with the visit of each vertex. All the labels thus generated are added to the DP graph before we start the solution process. Label Loading is illustrated on Figure 2. On this figure, $L_{i_1-\ldots-i_k}$ represents the label associated to partial path $\{v_{i_1}, \ldots, v_{i_k}\}$

The Meta Extension (ME) operator is used to obtain the complementary effect of Label Loading. While traversing each of the routes previously selected, we also add new *metavertices* to the original graph. These metavertices correspond to the remaining path from their associated original vertex to the destination depot and can be viewed as new depots. Every metavertex is defined as a successor of its associated orginal vertex. For instance, when traversing route $\{v_0, v_1, v_3, v_4, v_0\}$, the metavertex we associate with v_3 is equivalent to the remaining subpath $\{v_3, v_4, v_0\}$. This metavertex is added in the graph as a successor of v_3, as illustrated in Figure 3.

When a metavertex is added to the graph, we compute upper bounds on cost, time and load that allow deriving a feasible path with a negative reduced cost from it. To be extended to a metavertex,

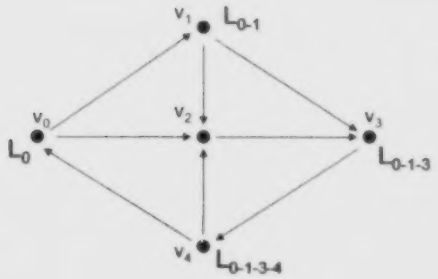

Figure 2: Label Loading: initial DP graph when $x_r > 0$ for route $r = \{v_0, v_1, v_3, v_4, v_0\}$.

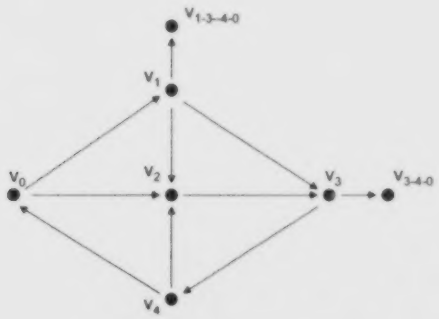

Figure 3: Meta Extension: metavertices when $x_r > 0$ for route $r = \{v_0, v_1, v_3, v_4, v_0\}$.

a label must respect these bounds. When this happens, we can obtain a complete route by simply checking that the path elementarity condition holds (i.e., that no vertex is visited more than once). The resulting label is then added to the set of depot labels as any other label extended to the depot.

Note that, with regard to the LDS policy, metavertices are considered as good neighbors, but are not counted in the good neighbor set.

The two techniques LL and ME, when used together, have the ability to rapidly identify small variants of the current MP solution routes. Operators like node insertion, node deletion, and path crossing (connecting the end of one route to the beginning of another) can be obtained with only a few label extensions. This permits to rapidly explore the neighborhood of the current MP solution. These techniques are very helpful to guide the search with the help of the global information provided by the current MP solution. They are very complementary with the LDS approach, which drives the search using local information (arc reduced costs).

2.3 Label Elimination

When the resource limits are not very constraining, a very large number of labels can be generated during the search for negative reduced costs paths. The motivation here is to determine (and remove) labels that can never be extended to the final depot with a negative cost.

What we propose is to compute a very simple lower bound on the costs of all paths that can be generated from a given label. This bound can be computed after each label extension in order to

establish whether the label can still yield a negative cost path or not. The bound has to be chosen to balance the quality of the information it provides with the time it takes to be computed. We propose to use again the two knapsack bounds previously defined for computing the maximal number of customers in routes. These two bounding schemes are changed to take only into account vertices that can still be added to the label. The cost of items (vertices) are set, in both cases, to be the cost of the cheapest outgoing arc minus the dual value associated with the vertex. For sake of efficiency, we only compute the linear relaxation of the bounds, which can be done in linear time. This is acceptable in the context of an Elementary Shortest Path algorithm where the elementarity condition already necessitates linear computations after each label extension. Once these bounds are computed, if the cost of the label plus one of the bound values is positive, the label is removed.

As this technique is expected to be useful essentially when the number of labels is large, we only trigger Label Elimination (LE) when the DISC parameter is at least 2.

Note that our lower bounds would not be valid when the elementarity condition is relaxed: in this case, new lower bounding schemes would be required in order to apply Label Elimination. Furthermore, these schemes should preferably be computed in constant time to avoid slowing down label extension. Hence, contrary to previous techniques (LDS, LL and ME), Label Elimination seems only convenient for the elementary path version of the column generation scheme. One should also note that, as we were performing this study, Lübbecke (2004) investigated the use of lower bounds in dynamic programming for column generation.

2.4 Other computational issues

As noted before, 2-path cuts stand out as a standard component of column generation schemes for vehicle routing problems. These cuts are very useful to improve the quality of the bounding scheme and to limit the number of nodes explored during the tree search. Although less essential when the ESPPRC-based lower bound is used, we have added this component to our implementation. 2-path cuts are generated at the root node of the search tree. They are implemented as described in Kohl *et al.* (1999) except for three points. First, TSPTW feasibility is checked using the ESPPRC solution module. This is simply done by giving high rewards (dual values) to the requested vertices and null rewards to the others. Second, we generate 2-path cuts each time that DISC (the LDS parameter) is greater than one, instead of waiting for the end of the column generation process. By doing this, we avoid having to solve repeatedly the last iterations of the process for which DISC is high and computing times are longer. Third, a limit is set on the total time spent for generating 2-path cuts. This limit is equal to 10% of the global time limit. When the total computing time spent for generating 2-path cuts reaches this limit, 2-path cuts are not searched for anymore. This limit avoid losing a lot of computing times in trying to solve some very difficult TSPTW instances. In return, some useful cuts might be forgotten.

It is also well-known that column generation methods often show very slow convergence due to heavy degeneracy problems. The LDS technique still increases the negative impact of this behavior, as it multiplies the number of iterations. In order to limit this number, we use a very simple stabilization method, Interior Point Stabilization, described in Rousseau *et al.* (2003). However, the impact of this method on computing times is rather slight since it mainly influences the first iterations of column generation and these iterations are very fast here. As a matter of fact, we stop stabilizing column generation as soon as 2-path cuts are generated or when the tree search begins.

A third noticeable issue concerns the handling of branchings. At each node of the search tree, we classically first try to branch on the number of vehicles. Unfortunately, limiting the number of vehicles sometimes leads to unfeasibility. We then check whether feasibility can be recovered easily by only generating new columns with the DISC parameter set to 0. We only branch on the number of vehicles when feasibility is recovered. Otherwise, or when the number of vehicles is not fractional, we select an arc for branching. We then proceed in the following way. The impact of the removal of every arc traversed a fractional number of times is evaluated by solving the updated Restricted Master Problem.

The arc whose deletion has the larger impact is selected. This policy permits to derive two branches for which the new constraint has an effective impact.

Finally, some implementation details need to be mentioned. During the subproblem solution phase, the first 50 columns extended to the destination depot with a negative cost are stored, without referring to dominance rules. Dominance rules are then activated for the remaining columns. This precaution avoids removing good columns when they are rare. The subproblem is stopped as soon as 500 columns of negative cost have been found, even if dominance rules have rejected many of them. We then attempt to complete the set of columns by extending every label of the DP graph towards the depot. At the Master Problem level, integrality of solutions is checked each time the simplex algorithm has been called for. This helps in finding good solutions quickly. The nodes of the search tree are treated in a best first order. This order is based on the value of the linear relaxation for their father node. This permits to maintain an increasing lower bound throughout the algorithm. When a linear program reaches this lower bound, one can then avoid triggering column generation uselessly.

3 Experimental Results

We have evaluated the performance of the proposed techniques on the well-known Solomon instances [17]. These instances are constituted of 3 types of geographical layouts. Customers are randomly located in problem sets r, clustered in problem sets c, and a mix of random and clustered structures in problem sets rc. Each type of instances is divided into two parts, the first part (r101-r112, c101-c109, rc101-rc108) having narrower time windows than the second part (r201-r211, c201-c208, rc201-rc208). Customer coordinates are identical for all instances within one type (i.e., r1,...,rc2): within one type, instances only differ with respect to the width of the time windows. Each instance contains 100 customers but smaller instances are created by considering only the first 25 or the first 50 customers. In the data sets, the distance matrix is not explicitly stated, but customer locations are given. Euclidean distances between these customers are calculated with one decimal point, to allow comparison with other published methods.

The computational study is divided into two parts. First, we evaluate the algorithm and compare its performance with existing column generation based solution schemes. Second, we select a representative subset of instances and evaluate more deeply the impact of the refinements.

Computational experiments were carried out on a 1.6 GHz processor with 256 Mb of RAM. The Master Problem was modeled and solved with Cplex 7.5 leaving all parameters to their default values. The maximum allowed time to find a solution was set to 3600 seconds.

3.1 Evaluation of our algorithm

Tables 1 to 6 show the efficiency of our algorithm. In these tables, LP bound and IP are respectively the value of the linear relaxation at the root node and the value of the optimal solution. Columns CPU, Iter and Col indicate the solution time, the number of subproblem calls and the number of generated columns. Column Cuts gives the number of 2-path cuts generated, while column Nodes represents the number of nodes explored in the search tree. Blank values indicate instances that cannot be solved within the imparted time.

With the time limit set to one hour, 124 instances are solved out of 168: all instances with 25 customers, 46 out of 56 instances with 50 customers, 22 out of 56 with 100 customers. These results compare very favorably with the results reported in the survey of Cordeau *et al.* (2001). In this survey, only 122 known optimal solutions are reported, compiled from four different papers, of whom many could only be solved with the help of massive computing ressources or time. Actually, 4 new instances are indicated to be solved in this paper, but they were recently dismissed by Chabrier (2003) and Irnich and Villeneuve (2003). As a matter of fact, these two recent papers improved significantly the state of the art.

Problem	LP bound	CPU	Iter	Col	Cuts	IP	CPU	Iter	Col	Nodes
r101-025	617.1	0.1	8	73	0	617.1	0.2	8	73	1
r102-025	546.3	0.2	10	150	0	547.1	0.4	16	171	3
r103-025	454.6	0.4	12	199	0	454.6	0.4	12	199	1
r104-025	416.9	0.5	14	255	0	416.9	0.5	14	255	1
r105-025	530.5	0.3	12	148	0	530.5	0.3	12	148	1
r106-025	465.4	0.7	28	293	6	465.4	0.7	28	293	1
r107-025	424.3	0.5	13	201	0	424.3	0.5	13	201	1
r108-025	396.8	1.2	25	305	0	397.3	2.2	38	572	3
r109-025	441.3	0.3	10	122	0	441.3	0.3	10	122	1
r110-025	438.5	0.6	19	282	1	444.1	3.2	87	642	14
r111-025	428.8	0.5	15	227	2	428.8	0.5	15	227	2
r112-025	387.1	0.7	11	356	0	393	7.3	78	1059	21
c101-025	191.3	0.3	7	228	0	191.3	0.3	7	228	1
c102-025	190.3	1.1	12	536	0	190.3	1.1	12	536	1
c103-025	190.3	1.5	14	595	0	190.3	1.5	14	595	1
c104-025	186.9	2.1	14	750	0	186.9	2.1	14	750	1
c105-025	191.3	0.5	11	281	0	191.3	0.6	11	281	1
c106-025	191.3	0.3	9	234	0	191.3	0.4	9	234	1
c107-025	191.3	0.3	8	213	0	191.3	0.3	8	213	1
c108-025	191.3	0.4	11	164	0	191.3	0.4	11	164	1
c109-025	191.3	1.9	26	397	0	191.3	1.9	26	397	1
rc101-025	406.6	0.6	15	242	0	461.1	7.6	249	865	121
rc102-025	351.8	0.4	9	186	0	351.8	0.4	9	186	1
rc103-025	332.8	0.5	11	184	0	332.8	0.5	11	184	1
rc104-025	306.6	0.6	8	119	0	306.6	0.6	8	119	1
rc105-025	411.3	0.5	15	209	0	411.3	0.5	15	209	1
rc106-025	345.5	0.5	16	193	0	345.5	0.5	16	193	1
rc107-025	298.3	0.5	8	248	0	298.3	0.6	8	248	1
rc108-025	294.5	1.4	12	246	0	294.5	1.4	12	246	1

Table 1: Optimal solutions Series 1: 25 customers

Problem	LP bound	CPU	Iter	Col	Cuts	IP	CPU	Iter	Col	Nodes
r201-025	460.1	0.7	13	247	0	463.3	0.8	26	314	3
r202-025	410.5	0.5	11	247	0	410.5	0.5	11	247	1
r203-025	391.4	1.5	15	538	0	391.4	1.5	15	538	1
r204-025	350.475	398.7	27	942	0	355	425.7	112	2073	12
r205-025	390.6	22.7	20	379	0	393	23.3	45	467	3
r206-025	373.6	1.3	19	369	0	374.4	3.5	55	665	5
r207-025	360.05	115.1	24	1011	0	361.6	117.6	46	1189	3
r208-025	328.2	9.5	23	1209	0	328.2	9.7	23	1209	1
r209-025	364.05	2.6	25	677	0	370.7	7.7	111	1217	9
r210-025	404.175	7.7	22	441	0	404.6	9.0	41	646	3
r211-025	341.327	4.8	25	822	0	350.9	784.1	264	12540	30
c201-025	214.7	0.4	10	160	0	214.7	0.4	10	160	1
c202-025	214.7	1.3	11	530	0	214.7	1.4	11	530	1
c203-025	214.7	9.1	17	778	0	214.7	9.1	17	778	1
c204-025	213.1	431.5	32	1028	0	213.1	431.6	32	1028	1
c205-025	214.7	1.2	15	473	0	214.7	1.2	15	473	1
c206-025	214.7	3.0	19	872	0	214.7	3.1	19	872	1
c207-025	214.5	2.7	15	682	0	214.5	2.8	15	682	1
c208-025	214.5	0.7	10	330	0	214.5	0.6	10	330	1
rc201-025	360.2	0.4	10	239	0	360.2	0.4	10	239	1
rc202-025	338	0.7	11	271	0	338	0.7	11	271	1
rc203-025	326.9	1.0	13	284	0	326.9	1.1	13	284	1
rc204-025	299.7	3.6	19	475	0	299.7	3.6	19	475	1
rc205-025	338	0.5	14	147	0	338	0.5	14	147	1
rc206-025	324	0.5	12	139	0	324	0.5	12	139	1
rc207-025	298.3	0.6	12	130	0	298.3	0.6	12	130	1
rc208-025	269.1	1057.9	30	891	0	269.1	1058.1	30	891	1

Table 2: Optimal solutions Series 2: 25 customers

Problem	LP bound	CPU	Iter	Col	Cuts	IP	CPU	Iter	Col	Nodes
r101-050	1044	0.7	20	324	2	1044	0.7	20	324	1
r102-050	909	1.9	29	438	0	909	1.9	29	438	1
r103-050	769.68	5.6	40	714	5	772.9	12.8	104	1064	12
r104-050	621.64	82.3	46	1182	14	625.4	422.3	205	3031	28
r105-050	893.65	1.6	30	523	4	899.3	4.8	98	733	14
r106-050	792.212	3.9	28	758	2	793	9.4	79	1446	7
r107-050	707.601	10.7	45	1088	7	711.1	43.6	194	1965	21
r108-050	598.475	214.3	64	1650	0					
r109-050	776.62	3.4	44	690	6	786.8	47.6	351	1548	80
r110-050	696.75	5.1	38	832	2	697	6.6	55	913	3
r111-050	697.285	6.3	41	963	8	707.2	129.1	346	2542	62
r112-050	616.77	23.5	51	1250	8	630.2	2286.7	1616	7716	436
c101-050	362.4	0.5	7	259	0	362.4	0.5	7	259	1
c102-050	361.4	1.7	18	353	0	361.4	1.7	18	353	1
c103-050	361.4	4.5	30	531	0	361.4	4.6	30	531	1
c104-050	358	2793.7	21	981	0	358	2793.8	21	981	1
c105-050	362.4	1.0	14	337	0	362.4	1.1	14	337	1
c106-050	362.4	0.9	14	332	0	362.4	1.0	14	332	1
c107-050	362.4	1.3	17	437	0	362.4	1.3	17	437	1
c108-050	362.4	1.2	19	262	0	362.4	1.2	19	262	1
c109-050	362.4	2.7	28	361	0	362.4	2.7	28	361	1
rc101-050	944	5.8	71	966	27	944	6.0	71	966	1
rc102-050	813.8	9.5	68	1272	11	822.5	234.9	714	4424	126
rc103-050	710.9	13.2	87	1534	2	710.9	13.5	87	1534	1
rc104-050	545.8	13.0	32	508	0	545.8	13.0	32	508	1
rc105-050	855.075	7.7	69	950	10	855.3	10.0	97	1204	3
rc106-050	719.996	11.2	71	945	4	723.2	31.7	197	1903	24
rc107-050	640.1	15.8	58	1027	1	642.7	27.3	107	1139	10
rc108-050	596.511	46.3	87	1526	2	598.1	175.7	236	2413	13

Table 3: Optimal solutions Series 1: 50 customers

Problem	LP bound	CPU	Iter	Col	Cuts	IP	CPU	Iter	Col	Nodes
r201-050	791.9	4.4	36	732	0	791.9	4.4	36	732	1
r202-050	698.5	12.1	39	1308	0	698.5	12.5	39	1308	1
r203-050	598.583	409.8	43	1566	0	605.3	543.0	119	2751	7
r204-050										
r205-050	682.85	18.1	62	1142	0	690.1	553.9	789	5544	100
r206-050	626.343	45.2	72	1886	0	632.4	1783.2	762	8807	82
r207-050	564.1	957.8	65	2585	0					
r208-050										
r209-050	599.825	422.9	63	1382	0	600.6	439.2	100	1680	3
r210-050	636.1	33.5	54	1791	0					
r211-050	528.62	1105.6	55	2271	0					
c201-050	360.2	3.0	28	479	0	360.2	3.1	28	479	1
c202-050	360.2	41.1	40	2214	0	360.2	44.2	40	2214	1
c203-050										
c204-050										
c205-050	359.8	9.9	36	1130	0	359.8	10.2	36	1130	1
c206-050	359.8	23.7	47	1687	0	359.8	25.4	47	1687	1
c207-050	359.6	34.8	23	1556	0	359.6	36.3	23	1556	1
c208-050	350.5	13.9	23	1076	0	350.5	14.2	23	1076	1
rc201-050	684.8	2.3	23	479	0	684.8	2.3	23	479	1
rc202-050	613.6	7.4	34	722	0	613.6	7.5	34	722	1
rc203-050	555.3	438.5	39	1090	0	555.3	438.7	39	1090	1
rc204-050										
rc205-050	630.2	6.2	38	670	0	630.2	6.2	38	670	1
rc206-050	610	3.6	26	500	0	610	3.7	26	500	1
rc207-050	558.6	21.8	33	686	0	558.6	21.9	33	686	1
rc208-050										

Table 4: Optimal solutions Series 2: 50 customers

Problem	LP bound	CPU	Iter	Col	Cuts	IP	CPU	Iter	Col	Nodes
r101-100	1634	9.1	52	1103	6	1637.7	11.7	75	1213	6
r102-100	1466.6	9.4	39	1151	0	1466.6	9.5	39	1151	1
r103-100	1206.83	51.1	82	2104	3	1208.7	166.7	172	2476	10
r104-100	957.258	1688.1	159	3356	0					
r105-100	1349.32	20.6	83	1537	18	1355.3	268.9	427	2949	58
r106-100	1228.1	46.5	90	2142	11	1234.6	1123.4	680	5204	88
r107-100	1054.8	307.5	140	2805	0					
r108-100										
r109-100	1135.06	50.1	104	2178	0					
r110-100	1055.98	110.8	113	2623	0					
r111-100	1034.89	103.4	114	2771	0					
r112-100	927.884	1639.4	125	2960	0					
c101-100	827.3	3.6	28	412	0	827.3	3.6	28	412	1
c102-100	827.3	14.7	47	1126	0	827.3	14.8	47	1126	1
c103-100	826.3	58.4	59	1718	0	826.3	58.8	59	1718	1
c104-100										
c105-100	827.3	6.2	35	681	0	827.3	6.2	35	681	1
c106-100	827.3	5.4	29	667	0	827.3	5.4	29	667	1
c107-100	827.3	7.2	40	683	0	827.3	7.2	40	683	1
c108-100	827.3	11.6	44	958	0	827.3	11.7	44	958	1
c109-100	827.3	25.3	60	1031	0	827.3	25.4	60	1031	1
rc101-100	1617.07	36.8	114	1820	47	1619.8	166.0	289	2884	28
rc102-100	1441.84	77.5	145	2464	0					
rc103-100	1245.36	231.2	198	2956	0					
rc104-100										
rc105-100	1509.8	69.5	142	2173	36	1513.7	100.0	196	2374	10
rc106-100	1343.08	137	143	2354	0					
rc107-100	1196.51	241.6	150	2609	0					
rc108-100	1105.1	1497.8	198	3623	0					

Table 5: Optimal solutions Series 1: 100 customers

Problem	LP bound	CPU	Iter	Col	Cuts	IP	CPU	Iter	Col	Nodes
r201-100	1140.3	473.6	125	2440	0	1143.2	1078.3	646	5623	54
r202-100	1022.24	802.3	109	4540	0					
r203-100										
r204-100										
r205-100	939.224	605.9	185	3919	0					
r206-100	866.868	3417.2	162	7128	0					
r207-100										
r208-100										
r209-100	841.402	1089.2	226	4904	0					
r210-100										
r211-100										
c201-100	589.1	36.7	73	1499	0	589.1	37.7	73	1499	1
c202-100										
c203-100										
c204-100										
c205-100	586.4	571.1	259	6174	0	586.4	596.2	259	6174	1
c206-100	586	190.3	112	3569	0	586	198.4	112	3569	1
c207-100	585.8	297.5	96	3891	0	585.8	307.1	96	3891	1
c208-100	585.8	308.2	131	3904	0	585.8	319.1	131	3904	1
rc201-100	1255.94	466.2	163	2473	0	1261.8	812.4	612	5289	38
rc202-100	1088.09	562.0	136	3337	0	1092.3	1383.3	475	6913	22
rc203-100										
rc204-100										
rc205-100	1147.61	514.5	145	2767	0	1154	1690.9	538	6675	20
rc206-100	1038.6	716.4	159	3672	0					
rc207-100	947.313	2787.9	161	4141	0					
rc208-100										

Table 6: Optimal solutions Series 2: 100 customers

Problem	LP bound	CPU	Cuts	IP	CPU	Iter	Col	Nodes
r210-050	636.1	38.1	0	645.6	4044	1056	8858	116
c203-050	359.8	6232.4	0	359.8	6237	33	2731	1
r109-100	1135.06	55.6	21	1146.9	15486	4619	14979	1054
c104-100	822.9	11063.6	0	822.9	11064	82	2101	1
rc102-100	1441.84	70.9	50	1457.4	7769	2126	8978	434
rc103-100	1245.36	230.8	45	1258	10119	1918	9674	260
rc107-100	1196.51	249.5	30	1207.8	11898	2580	10467	546

Table 7: Other optimal solutions with time limit set to 5 hours

As mentioned in section 1.3, the novelties of these two papers rely on their bounding schemes. Chabrier's algorithm (2003) uses the ESPPRC bound and is thus very similar to Feillet *et al.* (2004) and to the present algorithm before refinements. The method given in is able to solve 104 instances in less than hour, with computing conditions similar to the ones used here.

Irnich and Villeneuve (2003) evaluate the efficiency of forbidding k-cycles in the set of feasible routes. They assess their approach with the elimination of 2-cycles, 3-cycles or 4-cycles respectively. They propose this way a compromise between SPPRC and ESPPRC based bounding schemes. Indeed, using the ESPPRC based bound sometimes heavily penalizes the algorithm. This is especially noticeable for clustered instances, for which the non-elementary path bound directly provides the integer solution at the root node. Their approach permits the solution of respectively 111, 117 and 117 instances for $k = 2$, $k = 3$ and $k = 4$, in less than one hour on a Pentium 600 MHz. The behavior of their algorithm when 3- or 4- cycles are eliminated is globally comparable to ours, but relies on quite more complicated algorithmic structures. Also, our refinements are designed to be generic and could certainly be advantageously included in their algorithms.

Table 7 presents some complementary results obtained when the time limit is set to 5 hours instead of 1. This Table shows that some more instances can be solved. It is also interesting to know that our algorithm finds the optimal solution for many other clustered instances, without proving optimality in the imparted time.

Beside its efficiency, another interesting feature of our algorithm is its impact on the convergence of the column generation process. In all cases, the objective function decreases quickly toward the optimal value. In difficult instances, the intractabilities that remain lie in the last iterations of the process. Indeed, except LE, our refinements are not designed to be helpful in proving that no negative reduced cost column exists. Figure 4 illustrates the convergence of the column generation process (at the root node of the search tree) in the case of instance r103-100.

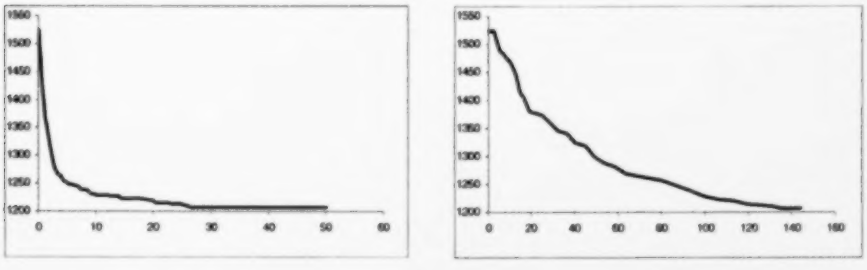

Figure 4: Convergence of column generation for r103-100, with or without refinements.

	without refinements			with all refinements		
Problem	CPU	Iter	Col	CPU	Iter	Col
r205-025	0.8	7	588	1.3	20	379
c203-025	153.7	15	3428	10.6	17	778
r104-050	82.7	17	2487	24.0	45	1131
r107-050	6.8	15	1484	6.2	33	939
r112-050	21.0	19	1955	9.7	40	1175
rc103-050	10.9	16	1156	8.1	37	889
rc106-050	2.5	14	933	4.5	42	692
rc107-050	12.8	16	1248	9.9	46	638
rc108-050	55.7	15	1393	20.6	33	797
r202-050	45.7	25	3797	13.6	39	1308
r205-050	18.5	21	2531	10.5	62	1142
r206-050	147.9	25	4920	39.2	72	1886
r210-050	145.9	25	4775	31.0	54	1791
c202-050	2596.6	61	13189	42.1	40	2214
c206-050	2482.5	75	15808	26.1	47	1687
c207-050	4672.6	68	17990	37.1	23	1556
c208-050	4619.3	83	20751	15.4	23	1076
rc207-050	146.7	20	2773	22.8	33	686
r103-100	162.2	38	4674	54.5	91	1938
r105-100	14.6	31	2338	17.7	75	1433
r106-100	95.7	38	3873	40.8	94	2015
r109-100	46.6	36	3236	39.3	85	2172
c102-100	476.1	31	2243	14.0	39	1110
c103-100	1636.2	33	4521	49.4	46	1705
c108-100	17.3	22	1472	12.5	44	958
c109-100	49.8	27	1477	27.4	60	1031
rc101-100	10.0	28	1956	18.0	75	1502
rc102-100	40.1	33	3389	31.4	89	1734
rc105-100	31.3	30	2600	27.2	83	1596
c201-100	1396.2	109	11379	96.6	105	2736
Total	19198.7	993	144369	761.5	1592	40694

Table 8: Global impact of the refinements

3.2 Detailed analysis of the impact of refinements

In this section, we propose some complementary numerical results, evaluating further the impact of the refinements. These results are presented in Table 8 and Table 9. For this purpose, a limited set of instances is used: the ones for which the LP relaxation is found between 10 and 100 seconds in Table 1 to Table 6. This criterion permits to select 30 instances with diverse characteristics. The LP relaxation is solved again for these instances using several versions of the algorithm. Since we focus on the refinements, in these tests we include neither the computation of 2-path cuts nor the tree search. We evaluate five solution schemes: a scheme without refinement, a scheme with all the refinements and three schemes including respectively each one of the three refinements alone. In each case, the tables present the CPU time needed for the solution and the total number of iterations and columns generated.

Some obvious conclusions can be drawn from these tables. First, the impact of the refinements is clarified in Table 8. In addition to the impressive speeding up of the solution process for difficult instances, the refinements permit to limit significantly the number of generated columns, though many more iterations are often needed.

	with LDS			with LL and ME			with LE		
Problem	CPU	Iter	Col	CPU	Iter	Col	CPU	Iter	Col
r205-025	1.7	24	442	1.1	11	464	0.8	7	588
c203-025	35.0	44	2288	29.7	12	1008	155.6	15	3428
r104-050	32.5	53	1574	45.8	24	1347	89.1	17	2487
r107-050	7.4	52	1232	6.2	21	1171	7.5	15	1484
r112-050	13.4	68	1597	18.1	29	1310	24.6	19	1955
rc103-050	7.5	46	1109	9.3	21	1084	11.1	16	1156
rc106-050	4.6	45	1108	3.1	22	760	3.0	14	933
rc107-050	13.3	54	1103	11.2	20	950	12.2	16	1248
rc108-050	24.4	57	1109	60.5	20	912	50.9	15	1393
r202-050	28.3	61	2754	12.4	23	1688	47.0	25	3797
r205-050	29.1	92	2504	8.1	28	1396	18.5	21	2531
r206-050	113.5	87	4440	25.6	29	1885	154.9	25	4920
r210-050	91.5	87	3650	55.1	38	2142	156.6	25	4775
c202-050	1730.5	179	12337	101.6	38	2564	2679.2	61	13189
c206-050	597.7	127	7358	34.7	36	1966	2528.5	75	15808
c207-050	695.6	115	7815	206.9	68	3559	4902.7	68	17990
c208-050	1253.8	139	9725	70.3	35	2356	4753.8	83	20751
rc207-050	28.3	50	1723	54.9	21	1617	157.0	20	2778
r103-100	74.7	125	3407	60.1	39	2103	178.1	38	4674
r105-100	34.9	116	3258	15.2	47	1626	15.8	31	2338
r106-100	72.5	128	3745	54.8	54	2496	105.8	38	3873
r109-100	67.7	138	3946	44.8	64	2248	51.9	36	3236
c102-100	37.0	96	2391	96.6	25	1677	493.4	31	2243
c103-100	74.5	73	2410	312.2	33	2822	1646.0	33	4521
c108-100	21.8	67	1299	11.4	21	818	21.4	22	1472
c109-100	42.0	67	1705	37.1	29	1103	63.2	27	1477
rc101-100	22.0	98	2505	12.2	45	1574	11.1	28	1956
rc102-100	43.4	103	3142	32.3	54	2289	46.8	33	3389
rc105-100	34.7	112	2971	28.0	56	1912	27.7	30	2600
c201-100	4596.5	579	16743	117.9	90	2472	1431.2	109	11379
Total	9829.8	3082	111390	1577.2	1053	51319	19845.4	993	144369

Table 9: Impact of the different refinements

Table 9 exhibits clearly the efficiency of Label Loading and Meta Extension. These combined techniques achieve a drastic improvement in terms of computing time and set of columns generated. On the opposite, their impact is very limited on the number of iterations.

The impact of LDS is more chaotic. It sometimes gives excellent results (e.g., instance c103-100), but in other cases it has a negative effect (e.g., c201-100). In most cases, it replaces some slow iterations with several quick ones and permits to limit the set of generated columns. Also, and that is the most interesting point, it is very complementary with LL and ME, and the best results are obtained when all three techniques are combined.

Finally, Label Elimination does not seem to have a prominent impact on the efficiency of the solution scheme. Introducing this component slightly deteriorate the solution times. Yet, further results show that many labels are eliminated: a mean of 7500000 labels are generated (over the 30 instances), a mean of 2500000 labels (a third of 7500000) are discarded. We can conclude that the loss in computing times is not offset by the elimination of labels. However, seeing that this issue was cautiously addressed in Lübbecke (2004) we have not investigated more deeply the behavior of this refinement.

4 Conclusion

In this paper, we have presented three techniques that accelerate column generation schemes for vehicle routing problems. We have applied these techniques to the VRPTW, where the computational experiments have demonstrated their important impact on the efficiency of the solution scheme. The Limited Discrepancy Search allows to rapidly execute the first iterations of column generation and to concentrate the effort on the last iterations. Label Loading and Meta Extension are simple techniques that prove very efficient, notably because they allow transforming the traditional label extension procedure into more powerful local search operators. Finally by computing a lower bound after each extension, we are able to identify and remove a large number of labels which can be proved worthless.

All these techniques together, but especially LDS, Label Loading and Meta Extension, have permitted the quick solution of many instances that could only be solved before with undue computing times. Furthermore, these techniques provide generic tools accelerating the convergence of column generation: good columns are found quickly and permit to converge quickly toward the optimal solution. The main difficulties then remain in the last iterations where interesting columns can be very difficult to find. Thereby, some specific questions like the generation of a good inital set of columns or the use of heuristic algorithms for trying to generate columns first, tend to become obsolete.

References

[1] Chabrier A. Vehicle routing problem with elementary shortest path based column generation. *Computers & Operations Research*, to appear.

[2] Cook W. and Rich J.L. A parallel cutting-plane algorithm for the vehicle routing problem with time windows. Technical Report TR99-04, Department of Computational and Applied Mathematics,Rice University, 1999.

[3] Cordeau, J.F., Desaulniers G., Desrosiers J., Solomon M.M., and Soumis F. The VRP with time windows. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, SIAM Monographs on Discrete Mathematics and Applications, pages 157–194. 2001.

[4] Desaulniers G., Desrosiers J., and Solomon M.M. Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys in Metaheuristics*, pages 309–324. Kluwer, 2001.

[5] Desrochers M., Desrosiers J., and Solomon M. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354, 1992.

[6] Desrosiers J., Dumas Y., Solomon M.M., and Soumis F. Time constrained routing and scheduling. In M.O. Ball, T.L. Magnanti, C.L. Monna, and G.I. Nemhauser, editors, *Network Routing*, Handbooks in Operations Research and Management Science, pages 35–139. Amsterdam, North-Holland, 1995.

[7] Feillet D., Dejax P., Gendreau M., and Gueguen C. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004.

[8] Gelinas S., Desrochers M., Desrosiers J., and Solomon M.M. A new branching strategy for time constrained routing problems with application to backhauling. *Annals of Operations Research*, 61:91–109, 1995.

[9] Harvey W. and Ginsberg M. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 607–615, Montréal, Canada, 1995. Morgan Kaufmann.

[10] Irnich S. The shortest path problem with k-cycle elimination (k≥3): Improving a branch and price algorithm for the VRPTW. In *TRISTAN IV*, volume 3, pages 571–574, 2001.

[11] Irnich S. and Villeneuve D. The shortest path problem with k-cycle elimination (k≥3): Improving a branch and price algorithm for the VRPTW. Technical Report G-2003-55, GERAD, 2003.

[12] Kallehauge B., Larsen J., and Madsen O.B.G. Lagrangean duality applied on vehicle routing with time windows - experimental results. Technical Report IMM-TR-2001-9, IMM, Technical University of Denmark, 2001.

[13] Kohl N., Desrosiers J., Madsen O.B.G., Solomon M.M., and Soumis F. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101–116, 1999.

[14] Lübbecke M.E. Dual variable based fathoming in dynamic programs for column generation. *European Journal of Operational Research*, 162(1):122–125, 2005.

[15] Rousseau L.-M., Gendreau M., and Feillet D. Interior point stabilization for column generation. Technical report, Centre de Recherche sur les Transports, 2003.

[16] Sol M. *Column generation techniques for pickup and delivery problems*. PhD thesis, Technische Universiteit Eindhoven, Netherlands, 1994.

[17] Solomon M.M. *Vehicle Routing and Scheduling with Time Window Constraints: Models and Algorithms*. PhD thesis, Department of Decision Sciences, University of Pennsylvania, 1983.